Tvorba klienta pro komunikační protokol Jabber

Creation of the client for the Jabber communication protocol

Bakalářská práce

Bachelor thesis

Petra Pešková

Vedoucí práce: Mgr. Jiří Pech, Ph.D.

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

2010

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracovala samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

V Českých Budějovicích dne

Anotace

Tato práce podává přehled a srovnání nejběžnějších komunikačních protokolů (ICQ, AIM, MSN, Gadu Gadu, IRC, Google chat a Yahoo! Messenger), podrobněji popisuje protokol Jabber. Součástí této práce je vytvoření vlastního klienta pro protokol Jabber.

Abstract

This Bachaleror's thesis describes and compares the best known communication protocols (ICQ, AIM, MSN, Gadu Gadu, IRC, Google chat a Yahoo! Messenger), especially the Jabber protocol. A part of this thesis is to design an instant messaging client based on Jabber protocol.

Poděkování

Ráda bych poděkovala panu Mgr. Jiřímu Pechovi, Ph.D. za laskavé vedení mé bakalářské práce a své rodině a blízkým za podporu.

Obsah

1 Úvod

Potřebu komunikovat mají lidé už od pradávna. Cíl, sdělit myšlenku, zůstává stále stejný, ale s časem se mění prostředky. V dnešní době, kdy má lidstvo k dispozici tolik prostředků, které zmenšují vzdálenosti mezi lidmi a šetří čas, mají lidé stále méně a méně času se s lidmi osobně setkávat. Proto je komunikace pomocí internetu stále častějším a důležitějším druhem komunikace. Pomocí internetu můžeme komunikovat mnoha způsoby, jedním z nejoblíbenějších je tzv. instant messaging.

Instant messaging – zkráceně IM je tolik oblíbený po právu. Jedná se totiž o velice rychlý a efektivní druh komunikaci. Komunikace probíhá v reálném čase, to znamená, že zprávy jsou doručovány téměř ihned po odeslání (prodleva je opravdu zanedbatelná, počítá se v řádu milisekund). Efektivnost spočívá především v zobrazení stavů kontaktů. Tím mám na mysli, že uživatel, který chce komunikovat, doslova vidí, kdo je z jeho přátel právě připravený komunikovat.

Tato práce popisuje nejběžnější protokoly, pomocí nichž komunikují lidé po celém světě. O způsobu výběru zmíněných protokolů pro svoji práci, píši v druhé kapitole. Tato kapitola dále popisuje, jaké prostředky byly zvoleny pro vývoj klienta. O vlastnostech jednotlivých protokolů se rozepisuji ve třetí kapitole. V následující čtvrté kapitole se nachází tabulky, které přehledně srovnávají vlastnosti všech zmiňovaných protokolů. Zvláštní pozornost je věnována protokolu Jabber, kterému je věnována celá pátá kapitola. Šestá kapitole je věnována popisu praktické části práce - implementaci zcela nového klienta, který je vytvořen jako součást této práce. V závěrečné kapitole se nachází zhodnocení celé práce.

2 Cíle práce a metodika

2.1 Cíle práce

Cílem této práce je podat přehled komunikačních protokolů, detailně popsat protokol Jabber a následně vytvořit vlastního klienta pro tento protokol, který nebude závislý na platformě.

2.2 Metodika

Kritéria pro výběr zmiňovaných protokolů

Parametrem při výběru protokolů zmiňovaných v této práci je jejich známost v České Republice.

Známost protokolů v ČR jsem hodnotila dle četnosti výskytu protokolů v česky psaných článků na internetu (viz tabulka č. 2.2.1 v příloze A) a dále podle počtu stažených klientů pro tyto protokoly (viz tabulka č. 2.2.2). Dle těchto kritérií jsem vybrala protokoly následujících komunikačních sítí: AIM, ICQ, MSN, Internet Reley Chat, Yahoo Messenger, Google chat a Gadu Gadu.

Kritéria vytvářeného klienta pro protokol Jabber

Klient by měl splňovat následující kritéria:

- měl by disponovat základními funkcemi komunikátoru (odesílání/přijímání textových zpráv, zobrazení stavu kontaktů v rosteru, modifikace vlastního statusu)
- jednoduchý a nenáročný
- nezávislý na platformě

Programovací jazyk

Při výběru programovacího jazyka jsem zvažovala zejména možnost práce s XML dokumenty a nezávislost na platformě.

Klient je psán v jazyce Java. Tento jazyk jsem si vybrala, především kvůli technologii JAXB, umožňující efektivně pracovat s XML dokumenty, je multiplatformový a má velmi kvalitní a přehlednou dokumentaci.

Vývojové prostředí

Program vznikl ve vývojovém prostředí NetBeans. Toto vývojové prostředí jsem zvolila, protože disponuje velmi kvalitním layout managerem a je tedy možné snadno vytvářet uživatelské grafické prostředí, je zdarma a mám s ním více zkušeností než s prostředím Eclipse, které jsem také zvažovala.

3 Přehled komunikačních protokolů

3.1 Pojem protokol

Pokud spolu chtějí komunikovat nějaké dva subjekty v síti, je především zapotřebí, aby byla jasně dána pravidla jejich komunikace, čímž se zabezpečí, že si budou oba subjekty rozumět. Soubor takových pravidel tvoří standart neboli protokol. Protokoly bývají často realizovány softwarově, ale jejich realizace může být i hardwarová či kombinovaná.

Výměna dat po síti probíhá odesíláním paketů (což jsou části dat v blocích) z jednoho uzlu v síti na druhý. Každý paket se skládá ze dvou částí, z přenášených dat a hlavičky – její formát protokoly specifikují. V hlavičce jsou obsaženy informace týkající se interpretace bitů, ze kterých se paket skládá.

Dále protokoly určují, jakým způsobem budou pakety zpracovávané při jejich odesílání nebo naopak přijímání, platné hodnoty jednotlivých bitů a postup pro jejich přijetí a také postupy pro ošetřování různých chyb, řízení toku, segmentaci, apod.

Komunikační protokol

Komunikační protokol pak specifikuje zejména pravidla, které udávají syntaxi a význam zpráv vyměňovaných mezi serverem a klientem. Jsou to zejména zprávy týkající se autorizace a registrace uživatele na server, změny statusu a samozřejmě textových zpráv mezi jednotlivými uživateli.

3.2 MSN

Název MSN čili Microsoft network označuje celou sadu internetových služeb poskytovaných společností Microsoft, posléze přejmenovanou na Windows Live. Pro IM služby slouží komunikační protokol **MSNP** (Microsoft Notification Protokol) a klient Windows Live Messenger, ten je volně ke stažení v 36 jazycích a je používán ve více než 50 zemích více než 320 milióny aktivních účtů každý měsíc. [1]

Historie

Spoustu verzí protokolu nebylo nikdy zveřejněno, jako například hned první verze MSNP1. Tuto verzi protokolu využíval oficiální, velmi jednoduchý klient MSN Messenger 1 spuštěný 22. 6. 1999. Dokázal zobrazit a modifikovat seznam kontaktů a posílat rychlé zprávy. Jako jediný komunikoval se službou AIM, což se stalo příčinou mnoha sporů a spolupráce se společností AOL byla přerušena. Ještě koncem téhož roku byla vydána další a verze protokolu s názvem MSNP2 a se zveřejněnou dokumentací [2].

Velkou změnu přinesla verze MSN8 a to ve způsobu autorizace. Ověřovací údaje se zasílaly na bezpečnostní server Microsoft Password (dnes známý pod názvem Microsoft Live ID). Kvůli novému zabezpečení Microsoft zablokoval všechny předešlé verze (což bylo v historii MSN poprvé, nikoliv však naposledy). Uživatelé se staršími klienty, se nemohli přihlásit ke svému účtu a byli nuceni klienta zaktualizovat. Další novinkou byla podpora hlasových služeb a web kamery.

V dalších verzích byly přidávány podpory datových zpráv (zprávy typu D, umožňující např. zobrazení uživatelských emoikon), přenosů souborů, integrace s Hotmail adresářem, zasílání offline zpráv, spolupráce s Yahoo! Messengerem a zvyšování kvality přenášeného hlasu a videa.

Přínosem nejnovější verze protokolu jsou nové funkce určené pro skupiny přátel. Příjemnou změnou je také přidání funkce MPOP (Multiple Points of Presence), umožňující přihlášení k jednomu účtu z více míst najednou.

Princip

Protokol se stará o výměnu zpráv mezi klienty, zjištění online stavu i s atributy, zjištění dostupných zdrojů (např.: mikrofon, web kamera). Klienti protokolu MSNP se během provozu připojují k několika serverům:

1. Dispatch server (DS) - první kontaktní místo pro klienta, poskytuje klientovi IP nejvhodnějšího notifikačního serveru. Po obdržení adresy se klient s tímto notifikačním serverem spojí.

2. Notification server (NS) – slouží k ověřování při přihlašování, odhlašování, zodpovídá za zobrazení přítomnosti a podporuje využívání SB. Po spojení klienta s NS proběhne autorizace, po jejím úspěšném dokončení se NS spojí s klienty všech uživatelů v listu kontaktů a zašle jim oznámení o aktuálním stavu uživatele.

3. Switchboard server (SB) – slouží k výměně zpráv mezi klienty, je třeba, aby oba klienti, jenž spolu komunikují, byli připojeni ke stejnému SB. K připojení k SB dochází následujícím způsobem: klient zahajující komunikaci s druhým klientem X, požádá svůj NS o přidělení nějakého SB pro komunikaci s X. NS mu ho přidělí a klien se s daným SB spojí. Poté se SB spojí s NS klienta X a pozve ho, aby se také připojil. Po připojení klienta X k samému SB může začít komunikace. Pokud klient chatuje ve více relacích najednou, je připojen k více SB najednou. K výměně souborů, videa a hlasu dochází prostřednictvím P2P spojení.

Celá komunikace spočívá v zasílání příkazů a zpráv. Příkazy se skládají ze 3 znaků identifikátoru a parametrů. Zpráva je specifický příkaz s identifikátorem MSG. Příklad obdrženého příkazu, pokud se odpojí uživatel ze seznamu kontaktů: `FLN name@hotmail.com`

3.3 IRC

Komunikace pomocí protokolu IRC je jednou z nejstarších internetových komunikací v reálném čase. Protokol vznikl vylepšením systému BBS (Bulletin Board System), které obstaral fin Jarkko Oikarinen. Největší změnou oproti BBS byla právě komunikace v reálném čase.

Historie

Protokol se poprvé začal používat v srpnu 1988 na serveru tolsun.oulu.fi finské univerzity v Oulu. Poté se začaly přidávat i další servery z ostatních finských univerzit a brzy se protokol začal užívat i na univerzitách v USA. Síť se začala velice rychle rozvíjet a během jednoho roku zahrnovala 40 serverů po celém světě. S rozrůstáním sítě začaly také první problémy. Není divu, IRC síť byla velice chaotická a anarchistická, na denním pořádku byly kolize nicků, nespojité připojení k síti a další problémy, bylo tedy jen otázkou času, kdy se někteří uživatelé odtrhnou a začnou zakládat své vlastní IRC sítě To se stalo hned v roce 1990, kdy se od sítě odtrhlo několik serverů, které založili síť A-net (Anarchy net), zbylá část sítě se pojmenovala EFnet (Eris Free Net). Další větších sítí byla síť Undernet vytvořené v roce 1992. Ačkoli byla vytvořena jen jako zkušební síť, získala si brzy přízeň mnoha uživatelů a to hlavně z důvodu ochrany jak uživatelů, tak i komunikačních kanálů. Ale i tato síť se rozdělila a to hned v roce 1994. Část zůstala stejná a z druhé části se zrodila síť DALnet. Ta zvedla nabídku služeb sítě na novou úroveň, nejdůležitější z nových funkcí bylo zavedení registrace nicků a zavedením G-lines (zákaz vstupu do IRC sítě některým uživatelům). V červenci 1996 se po dlouhých sporech rozdělila také původní síť Efnet. Oddělily se evropské servery (nově pojmenovaná síť IRCnet) a severoamerické servery (síť si ponechala původní jméno EFnet).

V dnešní době existují stovky dalších menších sítí, které vznikají, většinou s využitím upravené verze buď DALnet, EFnet, IRCnet, nebo Undernet sítě. Jedná se o středně velké sítě, specializované sítě i sítě regionální.

Princip

IRC protokol je podrobně popsán v dokumentu RFC 1459 [3] a je nadstavbou protokolu TCP. Zajišťuje tyto služby:

- doručování zpráv mezi serverem a klientem
- lokalizace jednotlivých klientů - server uchovává po celou dobu připojení klienta informaci o jeho umístění v síti
- správa kanálů

Síť je založena na modelu klient-server. Servery komunikují na portech 6660 – 6669. Struktura sítě vypadá tak, že každý server je zároveň centrálním bodem pro další servery (klienty) – servery v jedné síti tvoří acyklický graf. Z toho vyplývá, že spolu mohou komunikovat uživatelé různých serverů jedné IRC sítě Jakmile se klient se serverem spojí, obdrží unikátní devítimístné jméno, které si server uchovává. Všechny servery musí mít o klientech následující informace:

- skutečné jméno hostitele, na kterém je klient spuštěn
- uživatelské jméno klienta na hostiteli
- server, ke kterému je klient připojen

Pokud se přeruší spojení mezi servery jedné IRC sítě nastává tzv. **netsplit**. Takovým výpadkem může dojít k úplné separaci některých serverů, čímž se síť rozpadne na dvě nové. Uživatelé mohou dál komunikovat v rámci takto nově vzniklé sítě.

Uživatelé sítě spolu komunikují ve virtuálních místnostech – tzv. **kanály,** kde komunikují uživatelé, kteří mají něco společného. Název kanálu vždy

začíná znakem # (popř. &) a nesmí obsahovat mezery, čárky a být delší než-li 200 znaků. Uživatel může být zároveň v neomezeném počtu kanálů a vést více konverzací. V kanálu je možné vést buď hromadnou konverzaci nebo soukromý hovor. Založit nový kanál může kdokoli z uživatelů, tím se automaticky stává správcem tohoto kanálu (=channel founder) a jsou mu přiřazeny nejvyšší práva, těmi jsou například: vyhození uživatele z kanálu (popř. blokace přístupu přezdívky či IP adresy), změna režimu kanálu (volný vstup, vstup chráněný heslem, vstupem pouze na pozvání). Správce kanálu může zvolit další uživatele, kteří mu se správou budou pomáhat. Ti už mají jen částečná privilegia. Takový uživatel se nazývá channell operator a má znak @ umístěný vedle své přezdívky.

Zprávy

Komunikace mezi serverem a klientem probíhá výměnou zpráv, ne všechny zprávy generují odpověď. Zprávy obsahují příkazy (může jich až 15), jejich parametry a volitelně mohou obsahovat předponu.

Předpony využívají servery k označení původu zprávy. Pokud zpráva předponu obsahuje, je uvozena dvojtečkou, za kterou následuje předpona, pokud zpráva neobsahuje předponu, předpokládá se, že zpráva pochází ze spojení, ze kterého byla přijata. Pokud se v interní databázi serverů nenalezne zdroj, který předpona označuje nebo je zdroj registrován z jiného spoje, než z kterého zpráva přišla, pak server zprávu ignoruje.

Všechny příkazy používané v IRC protokolu se zapisují ve znakové sadě US-ASCII a uvozují se lomítkem, volitelně mohou mít parametry. Vedle povinných příkazů umožňujících spojení serverů/klientů, odesílání zpráv, manipulaci s kanály, zjištění stavu serveru/uživatele, existují i příkazy nepovinné – ty ale nemusí všichni servery/klienti podporovat.

IRC zprávy vždy končí znaky CR-LF (Carriage Return - Line Feed) a nesmí přesáhnout délku 512 znaků (a to včetně poslední povinné dvojice znaků), to

znamená, že maximální délka příkazu včetně jeho parametrů může být maximálně 510 znaků.

Komunikace

Pro zajištění bezpečnosti vzájemné komunikace klientů je požadováno, aby všechny servery (propojených do struktury stromu) byly schopni poslat zprávu právě jednou cestou ve stromu tak, aby v cestě nebyl žádný jiný klient. Zpráva je vždy doručena nejkratší cestou ve stromě. Klient může komunikovat různými způsoby, ve smyslu počtu příjemců jeho zprávy.

Cílovou skupinou příjemců může být jeden klient, ale mnohem častější je, pokud je příjemců více. Z toho je nejméně efektivní způsob, kdy uživatel napíše zprávu a udá seznam lidí, kterým má být zpráva doručena. Server pak kopie této zprávy pošle všem uživatelům ze seznamu. Při zasílání kopií zpráv server nekontroluje, jestli duplikáty neposílá stejnou cestou a posílá každou kopii zvlášť, v čemž tkví neefektivita tohoto způsobu.

Efektivnější způsob je, pokud se konverzace v kanále se zasílá pouze na ty servery, na kterých jsou uživatelé z daného kanálu. Pokud je v kanále více uživatelů stejného serveru, text zprávy se na server zašle pouze jednou. Server pak zprávu rozešle mezi všechny klienty, kteří se účastní konverzace v kanálu.

Poslední případ zasílání zpráv více příjemců, je pokud se zprávy zasílají uživatelům, kteří mají stejného hostitele či server (slouží pro informační zprávy daného serveru či poskytovatele) nebo zprávy, které se odesílají všem klientům či serverům v síti.

3.4 Yahoo! Messenger

YMSG - protokol Yahoo! Messengera je proprietární standard podporující zasílání zpráv, přenos souborů, videokonference i hlasový chat. Klient se serverem komunikuje na bázi TCP/IP spojení, standardně na portu 5050

(mohou být nastaveny i jiné, například pokud je tento blokován). Ne všechny služby nabízené klientem zajišťuje přímo protokol YMSG, například přenos souborů nejprve sjednán pomocí protokolu YMSG, ale o samotný přenos se stará protokol HTTP. Obdobně je to se službami webkamery. YSMG se stará o vyhledání a žádost o povolení přístupu na webkameru, ale přenos obrazu opět zajišťuje protokol HTTP. VoIP služby nezajišťují přímo Yahoo! servery, takže klient k této službě nemá přímé spojení.

Uživatelé spolu komunikují ve virtuálních místnostech. Každá místnost má své jméno, kapacitu cca 40 – 50 uživatelů a vždy patří do nějaké kategorie. Všechny tyto informace udržují Yahoo! servery, navíc každé místnosti přiřazují unikátní ID, které klient odesílá na server, pokud chce do místnosti vstoupit. Zprávy z místnosti jsou klientu přístupné, až po přihlášení do místnosti. Seznam kategorií a místností je klientu odesílán ve formátu XML.

Oficiální klient je poskytován pro operační systémy Windows a MacOS. Klienti jsou stále aktualizovány, ale ani nejnovější verze klienta není poskytována v českém jazyce.

Přihlašovací údaje se odesílají šifrovaně – pomocí metody digest-MD5. Po přihlášení server posílá seznam kontaktů i s jejich statusem. Poté je klient připraven k zahájení chatu.

Zprávy odesílané mezi uživateli obsahují 20 bytovou hlavičku, začínají ID relace, dále je v nich uvedena délka těla zprávy, příjemce, odesílatel, samotná zpráva a ukončení. Tyto řetězce jsou odděleny dvou-bytovým separátorem (\xc0 nebo \x80). Uživatel může měnit svůj status, stát se neviditelným, ignorovat určené uživatele a používat další možnosti, vše se provádí na základě zaslání konkrétního speciálního řetězce na server.

3.5 Protokol OSCAR

OSCAR (Open System for CommunicAtion in Realtime.) je komunikační protokol využívaný v systémech AIM a ICQ, produktů společnosti AOL. Protokol byl dlouhou dobu uzavřený. V roce 2002 byly na základě smlouvy zdrojové kódy protokolu OSKAR poskytnuty společnosti Apple Inc a tak mohli uživatelé protokolu OSCAR přímo komunikovat s uživateli iChatu. V roce 2006 uvolnil AOL SDK pro AIM, dovolil tak vývojářům použít plugin pro využití protokolu OSCAR v jejich klientech. Specifikace samotného protokolu byla vydána až v březnu 2008. Do té doby nebylo slovo Open v názvu protokolu zrovna na místě.

Klient, komunikující pomocí protokolu OSCAR, se během komunikace spojuje s několika servery. Základním serverem, ke kterému je klient připojen po celou dobu přihlášení uživatele, je Basic Oscar Service Server (BOSS). Stará se o směrování IM zpráv, notifikaci všech kontaktů a hlavní služby. Při ztrátě spojení s tímto serverem (odpojením) je uživatel offline. Avšak prvním serverem, ke kterému se klient připojí je Api.screenname server. Ten poskytuje veškeré autentizační služby, jako je například přihlášení uživatele. Dále je tu Api.oskar server poskytující přístup k různým webovým službám a pomáhající vyhledat vhodný BOSS server, ke kterému se klient má připojit. Poslední ze serverů, se kterými klient komunikuje je tzv. BART, neboli Buddy Art Server. BART poskytuje klientovi přístup pro stažení XML dokumentů, popisujících kontakty uživatele, obrázků a zvuků.

Binární protokol OSCAR využívá služeb TCP/IP protokolu pro síťové spojení. Všechny zprávy protokolu OSCAR jsou zapouzdřovány do tzv. FLAP framu. O toto zapouzdření paketů síťové vrstvy se stará FLAP (Frame LAyer Protocol) protokol. Hlavička framu je 6 bytová, která obsahuje neměnnou značku začátku framu, pole pro typ framu, číslo sekvence pro detekci chyb komunikace a velikost datové části framu. Datovou část framu, reprezentující zprávy protokolu OSCAR, tvoří komunikační jednotka SNAC.

Zprávy jsou rozděleny do různých skupin kategorií tzv. *foodgroups.* V každé *foodgroup* existuje více druhů zpráv. Mezi foodgroup patří skupiny: *OSERVICE (*obsahuje základní operace a datové typy, které jsou společné více serverům i skupinám – využíváno pokud klient žádá server o službu a pro vykonání akce je zapotřebí, aby se klient spojil s dalším serverem), *PD* (Permin/Deny - stará se o kontrolu povolení a zakázání nastavení uživatele), *LOCATE* (umožňuje zjištění a nastavení osobních funkcí, jako jsou různé uživatelské informace nebo obsah zpráv zasílaných v nepřítomnosti uživatele), *ICBM (Inter Client Basic Message* – zaměřená na zprávy zasílané mezi klienty), *INVITE*(pozvání uživatelů), *BUDDY* (slouží pro posílání oznámení klienta o stavu uživatelů v kontaktním listu), *FEEDBAG, BART* (systém umožňuje klientovi stažení různých BART položek u svých kontaktů i uživatele samotného, mezi tyto položky patří ikony pro kontakty, zvuk oznamující online stav kontaktu a další. Některé se stahují přímo ze systému BART, jiné jsou popsány pomocí XML souboru. Každá BART položka má své BART ID, což je 5 – 20 bytová binární hodnota).

OSCAR protokol používá velmi efektivní způsob kódování dat přenášených protokolem do organizované struktury a to pomocí datového typu TLV (Type Lenght Value). Tento datový typ se skládá ze tří polí – typ, délka a hodnoty.

16 bitové pole typ obsahuje číselný kód skupiny zprávy. Pole délka udává velikost dat v bytech a v posledním poli jsou samotná data, které se mají přenést.

3.5.1 ICQ

Síť ICQ vyvinula v roce 1996 čtveřice dvacetiletých mladíků z izraelské společnosti Miriabilis. Byli to pánové Yair Goldfinger, Arik Vardi, Sefi Vigiser a Amnon Amir. Tato společnost byla v roce 1998 prodána společnosti

AOL za cenu 407 milionů dolarů. Název této sítě vznik, jak je známo, z fonetické hříčky I Seek You („hledám tě“).

Oficiální klienty, vydávající společnost AOL, existují pro OS Windows, Macintosh a mobilní telefony. Nejnovější verzí klienta je ICQ 6.5. Podporuje češtinu a kromě zasílání IM zpráv podporuje i mnohé doplňkové služby. Těmi jsou zasílání SMS, video hovory, internetové volání, doplněk Xtraz (mini aplikace zobrazující nejrůznější informace, hry, seznamku …). Bohužel obsahují reklamu, jejíž blokování odporuje licenčním podmínkám stejně tak jako používání jiného než-li oficiálního klienta.

Jednotliví uživatelé jsou od sebe rozlišovány číslem UIN, unikátním číslem přiděleném systémem každému uživateli při registraci. Uživatel o sobě může vyplnit ještě další informace, jako je nick, jméno, adresa, zaměstnání, koníčky a mnohé jiné.Veškeré informace o uživateli, kromě UIN, je možno kdykoli změnit.

Síť ICQ je centralizovaná, hlavní centrální databáze se nachází na serveru www.icq.com a je propojená s dalšími servery. Centrální databázi umožňuje klientovi vyhledávání informace o dalších uživatelích dle různých parametrů (UIN, země původu, jméno, jazyk, atd.).

ICQ síť je v České Republice velmi využívaná - měsíčně 2,1 miliony uživateli, z toho 55% ji využívá denně. Počet celosvětových aktivních uživatelů je 32 milionů. Číselné hodnoty čerpané ze zdroje [4]. Ačkoli je v ČR takto oblíbená, je stejně často kritizována a to zejména kvůli licenčním podmínkám. Nejdiskutovanějšími body jsou: souhlas s anglickou verzí podmínek, souhlas, že licenční ujednání může být změněno a je na uživateli, aby si změny zjistil a souhlas s tím, že cokoli je posláno skrze síť ICQ se stává se majetkem společnosti AOL. Společnost AOL si vyhraňuje právo na změny protokolu, změny či (po)zastavení poskytovaných služeb a právo na prodej informací o uživatelích.

3.5.2 AIM

Název je zkratkou slov AOL Instant Messenger. Síť AIM je používaná především v USA. Oficiální klient podporuje OS Windows, Mac OS, iPhone Windows Mobile. Jeho počátky sahají do roku 1994, kdy na něm začal ve společnosti AOL pracovat pan Stephen D. William. Nejprve byl komunikátor užíván jen vnitropodnikově, ale v květnu 1997 byl k dispozici všem. AIM se postupně vyvíjel a ze systému, který původně umožňoval jen sledovat stav kontaktů a zasílání IM zpráv se stal systém podporující také videokonference. Poslední verze klienta AIM 7.0 podporuje textové zprávy, voice chat, video konference, emaily a doplňky jako jsou AOL rádio, přenos souborů, adresář, upomínky, vyhledávač a webový prohlížeč AOL, přeposílání zpráv i na mobilní telefon, spolupráci se sociální sítí Facebook či Twitter.

Ačkoli protokol OSCAR, na kterém je komunikátor postaven, byl ještě donedávna neotevřený, vydala už v minulosti společnost knihovnu pro implementaci klienta. Programátoři tak mohli vytvářet své vlastní klienty, ale s mnoha omezeními. V klientu musela být reklama a nesměl podporovat jiný protokol.

Uživatelé jsou identifikováni dle tzv. „screen name“ či emailové adresy ve tvaru uzivatel@aim.com. Screen name je řetězec písmen či číslic začínající písmenem.

3.6 Gadu-gadu

Jedná se o polskou síť, která byla poprvé spuštěna 17. 7. 2000 varšavskou společností Gadu-Gadu SA, která je financována z reklamy. V Polsku se stal Gadu-Gadu se zhruba 6 milióny uživateli nejrozšířenějším protokolem pro IM. (Dle polské agentury Megapanel PBI/Gemius byl v prosinci 2008 počet uživatelů Gagu-Gadu roven 5 778 532. [5]).

Gadu-Gadu je uzavřený protokol, existuje pro něj jeden oficiální klient, pojmenovaný stejně jako protokol a je určen výhradně pro OS Windows, ale také několik neoficiálních klientů i pro jiné platformy (např.: Pidgin, Miranda IM, Kopete a další). Nejnovější verzí oficiálního klienta je verze Gadu Gadu 10 (prozatím je dostupná jen beta verze). Podporuje různé skiny, obsahuje 150 smajlíků, podporuje avataty, přenos souborů, posílání SMS, kontrolu pravopisu v průběhu konverzace, VoIP, poslech internetového rádia Gadu Radio, sdílení fotek. Dále nabízí integraci se systémem MojaGeneracja.pl, tvoření skupin kontaktů, nastavení statusu, upomínky a po nabití kreditu i volání do polských sítí. Jistou nevýhodou je podpora jediného jazyka a to polštiny a hlavně fakt, že síť je zamořena velkým množstvím spamu.

3.7 Google Talk

Jedná se komunikační službu poskytovanou společností Google. Pro textovou komunikaci je využíván protokol XMPP (popis viz kapitola 4), hlasovou komunikaci a video chat zajišťuje protokol Jingle.

Pro přihlášení ke službě je třeba se přihlásit k serveru talk.google.com na portu 5222, Je vyžadováno zabezpečené spojeni TLS a jediný podporovaný šifrovací mechanismus při autentizaci je SASL PLAIN. Google talk vyžaduje registrací emailového účtu na serveru gmail.com. či googlemail.com. Příjemné je, že je služba Google Talk propojena s e-mailovou schránkou G-mail. Uživateli tedy stačí přihlásit se k jednomu z účtů a obdrží oznamování nových mailů i nových IM zpráv. Stejně tak je na každém účtu skladována historie mailů i konverzací.

Komunikovat je možné buď pomocí integrovaného klienta v poštovní schránce, pomocí alternativního klienta či klienta vyvinutého společností Google – Google Talk (či GTalk).

S aplikací Google Talk si mohou uživatelé kromě výměny textových zpráv, vyměňovat soubory, posílat hlasové zprávy (a to nejen mezi uživateli GTalk, je možné poslat hlasovou zprávu i mailem s přílohou mp3 souboru) a také umožňuje zobrazení právě přehrávané písně (spolupracuje s Windows Media Playerem a Winapem). Z alternativních klientů podporujících Google Chat zmiňuji klienty: Psi, Pidgin, Adium, Miranda, Kopete, Gajim a webový klient Meebo.

Protokol Jingle

Jingle je rozšíření protokolu XMPP, které vzniklo spoluprácí společnosti Google a XSF (XMPP Standards Foundation – nadace vzniklé z Jabber Software Foundation, zabývající se rozšířením XMPP). Toto rozšíření bylo vytvořeno, aby bylo umožněno vést a spravovat peer-to-peer spojení mezi jednotlivými subjekty protokolu XMPP. Což umožňuje efektivní přenos dat, v tomto případě se přenáší zejména video a hlas. Jingle umožňuje spravovat širokou škálu peer-to-peer relací a skládá se z několika modulů. Například video a voice chat zabezpečuje modul Jingle RTP Sessions, přenos souborů zajišťuje modul Jingle File Transfer, další sdílené aplikace zajišťuje modul Jingle XML Stream. Velkou výhodou protokolu je jednoduchá implementace do klientů protokolu XMPP.

Knihovna Libjingle

Libjingle je otevřená knihovna (vydána společností Google pod licencí BSD), podporující VoIP v síti Google Chat. První verze byla k dispozici v prosinci 2005. Knihovna obsahuje kód v jazyce C++, dokumentaci, popisující návrh aplikace, umožňující výměnu dat po síti a ukázkové aplikace.

4 Porovnání protokolů

Protokoly je možné porovnávat z několika různých úhlů. První tabulka pro přehlednost zobrazuje základní informace - jméno protokolu, síť, ve které je protokol používaný, typ licence a autora protokolu.

Tabulka 4.1 Základní informace o protokolech

Jméno protokolu	Komunikační síť	Licence	Autor
MSNP	MSN (Windows live)	uzavřená	Microsoft Corporation
IRC	IRC	otevřená	Jarkko Oikarinen
YMSG	YAHOO!Messenger	uzavřená	Yahoo! Inc.
OSCAR	ICQ, AIM	uzavřená	America online
Gadu-gadu	Gadu-gadu	uzavřená	Gadu-Gadu SA
XMPP	Google Talk, Jabber	otevřená	Jeremie Miller

Další tabulka srovnává klienty jednotlivých protokolů. V případě většiny protokolů se jedná o oficiální klienty, vydané společnostmi, které protokoly a celé komunikační sítě vlastní. V případě protokolu XMPP a IRC, u kterých neexistuje žádný jediný oficiální klient, byly porovnávány vlastnosti často užívaných klientů určených pro tyto protokoly. Mezi srovnávané vlastnosti jednotlivých klientů patří česká lokace programu a podporované platformy.

Tabulka 4.2 Srovnání klientů jednotlivých protokolů

Protokol	Klient	ČJ	Podporované platformy:				
			Windows	Mac	Linux	Mobil	Web
MSNP	WINDOWS LIVE MESSANGER	ANO	ANO	NE	NE	ANO [1)]	ANO
IRC	Např.: Gam, iIRC, Trillian, Meebo.com	ANO	ANO	ANO	ANO	ANO	ANO
YMSG	YAHOO! MESSENGER	NE	ANO	ANO	NE	ANO [2)]	ANO
OSCAR	ICQ	ANO	ANO	ANO	NE	ANO [3)]	ANO
	AIM	NE	ANO	ANO	NE	ANO [3)]	NE
GADU-GADU	GADU-GADU	NE	ANO	NE	NE	ANO	ANO
XMPP	Např. : Psi, Kopete, Meebo.com, QIP	ANO	ANO	ANO	ANO	ANO	ANO

[1)] jakékoliv zařízení umožňující prohlížení webu

[2)] BlackBerry, iPhone

[3)] Windows Mobile, iPhone

V další tabulce se nachází porovnání dovedností komunikačních sítí – jejich topologie, podpora dalších možností komunikace (kromě zasílání IM zpráv) či možnost přihlášení se ke svému účtu z více míst najednou.

Tabulka 4.3 Srovnání komunikačních sítí

Komunikační síť	Topologie	Přenos		Přihlášení z více míst najednou
		souborů	audio/video	
MSN (Windows live)	centralizovaná	ANO	ANO/ANO	ANO
IRC	decentralizovaná	ANO [1)]	NE/NE	NE
YAHOO	decentralizovaná	ANO [2)]	ANO/ANO [2)]	NE
ICQ, AIM	centralizovaná	ANO	ANO/ANO	NE
Gadu-gadu	centralizovaná	ANO	ANO/NE	NE
Google Talk, Jabber	decentralizovaná	ANO [3)]	ANO/ANO [3)]	ANO

1) o přenos se stará protokol DCC (Direct Client Connection) – P2P protokol užívaný v IRC sítí pro přenos souborů

2) samotný přenos se stará knihovna Jingle

3) přenos sjednává YMSG protokol, ale samotný přenos se uskuteční přes HTTP

Poslední tabulka srovnává protokoly ze síťového hlediska. Podává přehled portů, na kterých protokoly komunikují a podpory zabezpečeného spojení, popř. šifrování. Dále srovnává způsob identifikace jednotlivých uživatelů. Tady jasně z řady vybočuje IRC protokol, ten na rozdíl od všech ostatních neregistruje uživatele – ti jsou tedy úplně anonymní a mohou si libovolně měnit přezdívku, pod kterou vystupují.

Tabulka 4.4 Srovnání přístupu protokolů k síti

Komunikační síť	Port	Bezpečnost	Jednoznačná identifikace uživatele
MSN	1863	SSL	Windows live ID
IRC	6660-6669	SSL	přezdívka [1)]
YAHOO	5050	TLS + šifrovací mechanismus SASL-MD5	emailová adresa na jednom ze serverů yahoo.com, ymail.com, rocketmail.com, tzv. Yahoo ID
ICQ	4000	SSL	Devítimístné číslo tzv.: UIN
AIM	5190 – 3	TLS	emailová adresa ve tvaru uzivatel@aim.com nebo tzv. screen name
Gadu-gadu	443, 8074	-	číslo, udává se v pořadí registrace
Jabber	5222 - 3	TLS + jeden ze šifrovacích mechanismů: SASL-PLAIN, ANONYMOUS, DIGEST-MD5	JID ve tvaru name@server.com/resource za lomítkem udávaný zdroj je nepovinný
Google Talk	5222	TLS + šifrovací mechanismus SASL-PLAIN	Emailový účet na serveru Gmail

[1)] přezdívka je unikátní pouze v rámci jedné IRC sítě, uživatelé si jí neregistrují trvale (to umožňují pouze v některé IRC sítě)

5 Protokol XMPP

Proč tady zmiňuji protokol XMPP, když má být řeč o Jabberu? Tyto pojmy jsou často zaměňovány a mají k sobě velmi blízko. Všechno začalo pojmem Jabber, což bylo jméno projektu pana Jeremiho Millera. Projekt byl založen roku 1998 a šlo v něm o otevřenou komunikaci. Projekt zahrnoval server a sadu pravidel pro komunikaci – prastarý Jabber protokol. Protokol se stále vyvíjel a zlepšoval, až se z něj v roce 2004 stal standard s názvem XMPP. Takže Jabber dnes označuje komunikační platformu nebo také komunikační síť protokolu XMPP.

XMPP je zkratka slov Extensible Messaging and Presence Protocol, což by se dalo přeložit jako "rozšiřitelný protokol pro posílání zpráv a zobrazení stavu". Protokol je popsán v dokumentacích RFC3920 [6] a RFC 3921[7].

5.1 Princip komunikace

XMPP standard určuje pravidla komunikace mezi dvěmi koncovými body v síti. Jabber síť má architekturu klient-server, takže v případě využití protokolu ke komunikaci v síti Jabber budou koncovými body klient (kterého využívá uživatel pro komunikaci se serverem) a server (na kterém je uživatel zaregistrovaný). Veškerá komunikace probíhá pomocí výměny dat, která jsou ve formátu XML. Struktura XML dokumentů je definována pomocí XSD schémat (volně ke stažení např. na www.xmpp.org/ schemas/). Schémata jasně udávají, jak má XML dokument pro komunikaci vypadat. Změnou hodnoty jednotlivých XML elementů se změní druh požadavku či odpovědi na požadavek.

Pro zahájení komunikace je nejprve nutno zahájit spojení mezi serverem a klientem. Klient se připojuje na server nejčastěji TCP/IP spojením, ve kterém

se často zahajuje komunikace pomocí zabezpečeného TSL spojení. Poté jsou založeny streamy, ve kterých se zasílají fragmenty XML dokumentů. Protože je stream jednostranný tok dat, je zapotřebí, aby byly založeny dva streamy. V jednom posílá klient data na server a ve druhém komunikuje server s klientem. Konverzace tedy probíhá ve dvou XML dokumentech, které se mění v průběhu času.

Celá konverzace tedy vypadá zjednodušeně takto:

- klient naváže síťové spojení se serverem (ten poslouchá standardně na portu 5222)
- klient otevře stream
- server klientu odpoví – otevře svůj stream
- poté ve streamech probíhá výměna XML elementů, které zajišťují autorizaci, zasílání zpráv a všechny ostatní požadavky i odpovědi na ně. Tyto elementy tvoří XML fragmenty, které se přenášejí ve streamu a dohromady pak tvoří XML element
- ukončí se streamy
- ukončí se spojení se serverem

5.2 XML Stanzy

XML stanza je jeden samostatný celek, který si ve streamu vyměňují klient a server v rámci jednoho uceleného fragmentu XML. Tento celek tvoří jeden nebo více elementů. Stanza vždy začíná úvodním tagem, který je párový. Protože je stanza přímý potomek kořenového elementu <stream />, je vždy úvodní tag (i jeho uzavírací tag) v hloubce 1 XML streamu. Stanza se upřesňuje atributy a může obsahovat i další elementy – ty se poté nazývají subelementy.

V protokolu XMPP existují 3 základní stanzy, jsou to:

- *<iq />* používá se pro zasílání požadavků a odpovědí na ně
- *<message />* pro práci se zprávami
- *<presence />* pro práci se statusem

Každá stanza má také atributy, společnými atributy jsou:

- *to* - určuje příjemce
- *from* – určuje odesílatele
- *id* – nepovinný, používaný hlavně pro stanzu iq, určená pro identifikaci požadavku
- *type* – specifikuje daný element, u všech stanz je společná možná hodnotou error
- *xml:lang* – definuje jazyk zpráv

5.2.1 Stanza <iq />

Název stanzy Iq je zkratka slov Info/Query. Stanza je určená pro zasílání žádostí na server. Tyto žádosti mají strukturu požadavek – odpověď. Tato struktura je závazná, není požadavku bez odpovědi naopak. Aby bylo možné tuto strukturu dodržovat, je nutné, aby každý požadavek obsahoval atribut id. Odpověď na konkrétní požadavek bude mít stejné id. Z toho plyne, že id identifikuje celou žádost, což je pár: požadavek a odpověď na něj. Je třeba, aby každé id ve streamu bylo unikátní. Existuje více druhů požadavků i jejich odpovědí. Požadavky mohou buď nastavovat různé údaje, nebo mohou chtít tyto údaje od serveru získat. Odpovědi buď potvrzují úspěšné zpracování požadavku, nebo oznamují chybu. Aby bylo možné rozeznat požadavek od odpovědi i druh požadavku či odpovědi, má stanza další povinný atribut type.

Jedná-li se o zaslání požadavku, může atribut *type* nabývat jednu z hodnot:

- *get* – požadavek chce získat nějaká data uložená na serveru
- *set* – požadavek nastavuje data uložená na serveru na nové hodnoty, které jsou v požadavku obsaženy

Jedná-li se o odpověď serveru na požadavek, nabývá atribut *type* jedné z hodnot:

- *error* – nepodařilo se úspěšně zpracovat požadavek
- *result* – podařilo se zpracovat požadavek správně, v případě odpovědi na požadavek typu get, obasahuje i data, které klient žádal

5.2.2 Stanza <Message />

Jak napovídá název, slouží tato stanza pro zasílání zpráv mezi uživateli, což je hlavní účel protokolu XMPP. Pravidla doručování zpráv jsou popsány v článku Server Rules for Handling XML Stanzas (RFC 3921, odstavec 11). Ve zkratce tyto pravidla říkají, že server, na který je přihlášený klient odesílající zprávu, zkontroluje, na jaký server má být zpráva odeslána. Pokud se jedná o tentýž server, postará se sám o doručení zprávy, pokud je příjemce zprávy přihlášen k jinému serveru, předá na tento server zprávu a o doručení se dál nestará.

Stanza `<message />` funguje jako kontainer, který vyžaduje určitou formu. Tuto formu mu dávají atributy. Tento „kontainer“ slouží jako jakýsi obal pro přenášené informace (což je samotná zpráva, příjemce, odesílatel a další). Tyto informace jsou uchovávány v sublementech (tzv. child elementy elementu message).

Atributy:

- **type** – nepovinný atribut, který příjemci zprávy oznamuje jakého druhu je příchozí zpráva (klient může reagovat odlišně na různé druhy zpráv). Atribut *type* může nabývat následujících hodnot:
  - *normal* – používá se u jednoduchých zpráv, u kterých se nijak obzvláště neočekává odpověď či vyvolání diskuse. Tento typ je nastaven jako výchozí, tzn.: není-li typ určen, zasílá se zpráva typu *normal*
  - *chat* – používá se při posílání zprávy, která je částí konverzace. Pro identifikaci vlákna konverzace slouží subelement `<thread />` (viz dále).
  - *groupchat* – používá se pro upozornění klienta, že zpráva nepochází od uživatele, ale z komunikační místnosti
  - *headline* – používá se převážně u zpráv vytvářených automatickou službou (RSS čtečky, různé informace od sportu po ekonomii), na tyto zprávy se neodpovídá
  - *error* – používá se, pokud se jedná o chybovou hlášku zasílanou serverem. Chyby mohou nastat z mnoha důvodů. V subelementu `<error />` je uvedeno číslo chyby (viz dále).
- **from** - označuje JID odesílatele zprávy (buď se jedná o JID uživatele či o JID místnosti – pokud se jedná o skupinovou konverzaci)
- **to** – udává příjemce zprávy, tím může být bud uživatel, (*JID ve tvaru uzivatelske_jmeno@server + volitelně /zdroj*) či server (JID obsahuje jen server). *To* atribut je nepovinný, pokud není uveden, tak se zpráva vrátí zpět odesílateli či serveru (dle okolností)

- **id** – pokud je na zprávu očekávána odpověď je vhodné použít identifikátor (vhodné – ne nutné, jde opět o volitelný atribut). Je to z toho důvodu, že odpověď se vytváří tak, že se vymění atributy *from* s *to*. *Id* tedy zůstává a lze z něj přesně poznat, na kterou zprávu bylo odpovídáno.

Subelementy

Jak je popsáno výše, subelementy uchovávají a popisují různé informace o zprávě. Všechny jsou nepovinné.

- **<body />**

  V tomto subelementu se uchovává samotný text zprávy.

- **<subject />**

  Používá se pro nastavení předmětu zprávy. U zpráv typu chat se spíše nepoužívá. Užívanější je u zpráv typu normal,kde může být subjekt zobrazen ve stylu listu čí inbox položek a zejména pak u groupchatu pro zobrazení témata.

- **<error />**

  V tomto subelementu se přenáší informace o vzniklé chybě, ty mohou být buď standardní (stejné jako u protokolu HTTP – popsány v RFC 2616, odstavec 10[8]) či může být zadán vlastní seznam chyb. Informace o chybě se skládá z čísla a textového popisu chyby. Chyba nastane, pokud uživatel odešle zprávu, ale server zprávu nedokáže doručit. Takovou zprávu zašle server zpět odesílateli s přidaným subelementem error.

- **Subelement <thread />**

  Nese sebou informaci, která klientům umožní seskupit části konverzace tak, aby dali dohromady celý rozhovor. Má stále stejnou hodnotu v celém vlákně dané diskuse. Nesmí nastat případ, aby mělo více různých

rozhovorů uskutečněných v jednom streamu stejnou hodnotu subelementu `<thread />`.

- **Subelement <x />**

  Subelement `<x />` poskytuje jakýsi kotvící bod pro informace, které mají být ke zprávě strukturovaně připojeny. Je třeba, aby měl subelement `<x />` vždy definovaný jmenný prostor.

  Příklad ukazuje element message s připojením URL:

```
<message type='headline' from='news@ABC-company.com'
to='john@ABC-company.com'>
     <x xmlns='jabber:x:oob'>
     <url>http://www.abc-company.com/news/action</url>
<desc> Pozvánka na akci.</desc>
    </x>
</message>
```

  V příkladě byl jmenný prostor subelementu <x /> jabber:x:oob, který umožnil připojit přílohu URL. Existuje samozřejmě více jmenných prostorů subelementu <x /> a kdokoli si může definovat svůj vlastní pro připojení přílohy, jaká ho napadne. Tyto jmenné prostory se definují v JEP (= rozšířené protokoly, Jabber Enhancement Proposals).

- **Subelement <html />**

  Tento subelement podporuje formátování zpráv v XHTML jazyce. Je zapotřebí, aby byl v `<html />` byl definován jmenný prostor: 'http://jabber.org/protocol/xhtml-im' a v `<body />`, které nese takto formátovaný text, jmenný prostor: 'http://www.w3.org/1999/xhtml'.

  Dále je třeba splnit podmínku, že pokud `<body />` obsahuje takto formátovaný text, je zapotřebí, aby byl v elementu `<message />`, subelement `<body />` zopakován ještě jednou a to s klasickým, neformátovaným textem.

Příklad formátované zprávy:

```
<message to=" " type=" ">
 <html xmlns='http://jabber.org/protocol/xhtml-im>
 <body xmlns='http://www.w3.org/1999/xhtml'>
<span style="font-family:Arial; font-size:10pt,
font-color:green"> Pojď do lesa</span>
 </body>
 </html>
 <body>Pojď do lesa</body>
</message>
```

5.2.3 Stanza <Presence />

Existují dva účely stanzy `<presence />`. Tím prvním je zprostředkování přenosu informací o stavu uživatele – jeho přítomnosti, aktivitě či ochotě uživatele odpovídat na zprávy. Tyto informace nejsou určeny všem, ale pouze vybraným uživatelům. A tím se dostáváme k druhému účelu, tím je nastavení oprávnění dostávat informace o změnách stavu uživatele.

Nejzákladnější rozdělení stavu uživatele je, zda je uživatel dostupný nebo nedostupný. Pokud je dostupný, mohou mu být ihned doručovány zprávy, pokud je nedostupný, tak se zprávy uchovávají na serveru, aby mohly být doručeny, až se uživatel připojí.

Atributy

Stanza `<presence />` může mít volitelné atributy: *type, id, from* a *to*.

Jak je psáno výše, atribut type je nepovinný. Dle jeho přítomnosti lze rozpoznat, k jakému účelu stanza slouží. Pokud není přítomen, slouží `<presence />` k prvnímu zmiňovanému účelu – oznámení o stavu uživatele. Pokud je atribut type přítomen, slouží stanza `<presence />` k žádosti o zasílání/nezasílání presence uživateli, informacích o schválení čí zamítnutí těchto žádostí, pro chybové hláška apod.

Atribut *type* může nabývat následujících hodnot:

- **unavailable** – subjekt není k dispozici pro komunikaci
- **subscribe** – žádost o povolení zasílání informací o dostupnosti
- **subscribed** – povolení příjmu informací o dostupnosti
- **unsubscribe** – žádost o zrušení zasílání informací o dostupnosti
- **unsubscribed** – zrušení povolení/zamítnutí žádosti o zasílání informací o dostupnosti
- **probe** – žádost o aktuální stav subjektu, může vytvořit jen server a to na žádost uživatele
- **error** – chyby se týkají procesu doručení naposledy zaslané stanzy presence

Atributy *from, to* a *id* slouží pro obdobné účely jako u ostatních stanz (viz společné vlastnosti). Id se u stanzy `<presence />`většinou nepoužívá, vzhledem k tomu, že oznámení přítomnosti je jen jednocestná záležitost, bylo by zbytečné, aby se stanza specifikovala tímto atributem.

Subelementy

Subelementy stanzy `<presence />` slouží k bližšímu určení stavu dostupnosti uživatele, z toho plyne, že mohou být použity za předpokladu, pokud stanza není specifikovaná atributem *type*. Dále stanza nesmí obsahovat více stejných subelementů a žádný z nich nesmí obsahovat žádný atribut.

- **<show />**

Upřesňuje informace o dostupnosti uživatele. Jedná se o volitelný subelement, pokud není uveden, považuje se uživatel za dostupný online).

Pokud je subelement `<show />` uveden, musí nabývat jedné z následujících hodnot:

- **away** – uživatel není dočasně k dispozici
- **chat** – uživatel by si rád popovídal
- **dnd** – zkratka z Do Not Disturb , uživatel si nepřeje být rušen
- **xa** - zkratka z eXtended Away , uživatel není delší dobu k dispozici

- **<status />**

  Popisuje stav uživatele ještě blíže než předchozí subelement. Na rozdíl od něj, nenabývá žádných předem stanovených hodnot, ale nese s sebou uživatelem zadaný text, popisující jeho aktivitu.

- **<priority />**

  Vzhledem k tomu, že uživatel může být ke svému účtu přihlášen najednou z více míst (zdrojů) najednou je třeba rozlišit, jaký zdroj bude udávat uživatelovo dostupnost. Je to ten zdroj, který má aktuálně nejvyšší prioritu. Hodnota musí být celé číslo v rozsahu – 128 a + 127. Při absenci subelementu je výchozí hodnotu 0.

6 Implementace

6.1 Popis programu

Tento klient je určený lidem, kteří chtějí komunikovat prostřednictvím protokolu Jabber. Klient umožňuje:

- připojit se k serveru, kde je uživatel zaregistrovaný a přihlásit uživatele
- posílat a přijímat zprávy od ostatních uživatelů
- odesílat informace o svém aktuálním statusu a tyto informace od ostatních uživatelů přijímat, zpracovat a zobrazit
- získat ze serveru seznam kontaktů a zobrazit jej

6.2 Objektový návrh

Jak již bylo v této prácí zmíněno, komunikace mezi serverem a klientem probíhá ve dvou streamech pomocí výměny fragmentů XML dokumentu. Struktura těchto XML dokumentů je definována pomocí XSD schémat, které je možné získat například na webu http://www.XMPP.org. A tím se dostáváme k jádru pudla celého programu – využití technologie JAXB. Jedná se J2EE technologii, která slouží pro převod XML na Java objekty a naopak – viz dále.

Ze schémat jsem JAXB kompilátorem xjc vygenerovala Java třídy a to následujícím postupem:

V příkazovém řádku je třeba zadat cestu ke kompilátoru xjc a následně použití příkaz xjc, jehož parametr je cesta k souboru, jenž se má zkompilovat.

Např.:

```
C:\Program Files\Java\jdk1.6.0_16\bin>xjc "C:\schema.xsd"
```

Tyto vygenerované třídy reprezentují XML dokument tak, že každý element XML je objektem v Java třídě. Následně je využíván nástroj unmarshalling, jenž se stará o převod XML dokumentu do instancí objektů tříd Javy. Tyto objekty, jsou posléze měněny pomocí metod (základní metody typu get/set jsou generovány kompilátorem). Tím se ve finále změní XML dokument, který z objektů získáme inverzním nástrojem marshalling. Marshalling obecně znamená transformaci paměťové reprezentace objektu do formátu vhodného k přenosu. V tomto konkrétním případě je onou paměťovou reprezentací instance třídy obsahující objekty, které představující element XML dokumentu a formátem vhodným k přenosu je samozřejmě XML.

Tolik k části programu, kterou tvoří generovaný kód. Následuje vlastní tvorba kódu.

Jádrem celého programu je abstraktní třída AbstractXMPPObject.java, která uchovává společné proměnné základních tříd a deklaruje metody, které implementují tyto základní třídy (těmi jsou Roster.java, Login.java, Messanger.java a Presence.java). V této abstraktní třídě se nachází deklarace stěžejní metody dataReceived(Object data), určené pro příjem dat ze serveru. V implementaci této metody u každého potomka abstraktní třídy je test, který zjišťuje, jestli jsou data obdržené ze serveru určené právě této třídě. Pokud ano, třída reaguje zpracováním přijatých dat. Toto testování je vidět v následujícím příkladě na řádce č. 3.

Příklad implementace metody dataReceived(Object data) ve třídě Messenger.java:

```
1 @Override
2 public void dataReceived(Object data) throws
  LoginXMPPException {
3       if(data instanceof Message{
4               Message m = (Message) data;
5               ZPRACOVÁNÍ PŘÍCHOZÍ ZPRÁVY
6       }
7 }
```

6.3 Chování programu

Po spuštění programu se objeví přihlašovací okno, do kterého je třeba, aby uživatel vepsal své přihlašovací údaje. Přihlášení proběhne po stisku tlačítka přihlásit.

Po úspěšné autentizaci uživatele se rozešle eventa LoginComplete, na kterou reaguje jednak třída Presence.java (odešle oznámení o stavu on-line) a dále třída Roster.java (zažádá server o poskytnutí seznamu kontaktů). Uživateli se zobrazí hlavní okno aplikace, kde vidí seznam kontaktů. Stav jednotlivých kontaktů indikuje ikona umístěná vlevo od jména kontaktu. Rozšiřující zprávava statusu se zobrazuje v tooltipu po najetí myši na kontakt. Otevření okna pro zaslání zprávy proběhne po dvojkliku na kontakt v rosteru, kterému bude zpráva adresována. Změnit aktuální stav může uživatel vybráním jedné z voleb v menu Stav.

6.3.1 Navázání spojení se serverem, autentizace

Aby mohl začít klient pracovat, musí se nejprve spojit se serverem. Spojení se serverem se naváže na bázi TCP/IP spojení na standardním portu 5222 (zařídí metoda connectNormal() ze třídy Connection.java). Po navázání spojení začne klient se severem komunikovat. Komunikace začíná zasláním hlavičky a založením streamu pro zasílání dat směrem od klienta na server. Klient zašle:

XML element:	Kód programu:

`<?xml version=\"1.0\"?>`
`<stream:stream xmlns='jabber:client'`
`xmlns:stream='http://etherx.jabber.o`
`rg/streams' to='example.com'`
`version='1.0'>`
	metoda:
	sendRawHeader(String to)
	ze třídy DataSender.java

na to server reaguje založením streamu pro zasílání dat ze serveru a odpoví:

```
<stream:stream xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams' id='123'
from='example.com' version='1.0'>
```

poté ještě server zašle seznam funkcí (= features), které server podporuje.

```
<stream:features>
      <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
               <required/>
      </starttls>
</stream:features>
```

V tomto konkrétním příkladu server oznamuje, že podporuje zabezpečené spojení. Pokud tedy klient bude chtít komunikovat zabezpečeně, je třeba, aby vyčkal, než mu server zašle seznam features a poté na ně reagoval. Seznam všech oficiálních funkcí ukazuje tabulka.

Tabulka 5.1 Seznam features

Funkce	XML Element	Popis
Amp	`<amp xmlns='http://jabber.org/features/amp'>`	Podpora pokročilého zpracování zpráv (Advanced Message Processing)
Compress	`<compression xmlns='http://jabber.org/features/compress'>`	Podpora komprese streamu
Iq-auth	`<auth xmlns='http://jabber.org/features/iq-auth'>`	Podpora nezabezpečené autentizace (Non-SASL)
Iq-register	`<register xmlns='http://jabber.org/features/iq-register'>`	Podpora In-Band Registrace
Bind	`<bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>`	Podpora resource binding – zlepšení definice zdroje vázaného na zařízení
Mechanisms	`<mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>`	Podpora SASL ověření
Session	`<session xmlns='urn:ietf:params:xml:ns:xmpp-session'>`	Podpora vytvoření relace pro chat
Starals	`<starals xmlns='urn:ietf:params:xml:ns:xmpp-tls'>`	Podpora zabezpečeného spojení TLS
Dialback	`<dialback xmlns='urn:xmpp:features:dialback'>`	Podpora server dialback (ověření identity mezi servery)
Sm	`<sm xmlns='urn:xmpp:sm:2'>`	Podpora aktivního řízení streamu

Tabulka inspirována z [9]

Klient oznámí serveru, že bude komunikovat zabezpečeně:

XML element:

```
<starals
xmlns='urn:ietf:params:xml:ns:
xmpp-tls'/>
```

Kód programu:

Metoda: dataReceived(Object data) - step 0

ze třídy TLSDigestMD5Login.java

Server potvrzuje:

```
<proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

Začíná zahájení zabezpečeného spojení. Využívá se aktuálního spojení, které bylo navázáno dle protokolu TCP/IP, ve kterém se vytvoří kanál pro zabezpečené spojení.

Zahájení nového streamu. Pro zabezpečenou komunikaci směrem od klienta k serveru.

XML element:

```
<stream:stream
xmlns='jabber:client'
xmlns:stream='http://etherx.
jabber.org/streams'
to='example.com' version='1.0'>
```

Kód programu:

Metoda: dataReceived(Object data) - step 1

ze třídy TLSDigestMD5Login.java

Server také založí stream, aby mohla probíhat komunikace i ve směru od serveru k uživateli a zašle seznam podporovaných funkcí.

```
<stream:stream
     xmlns='jabber:client'
     xmlns:stream='http://etherx.jabber.org/streams'
     from='example.com'
     id='c2s_234'
     version='1.0'>
<stream:features>
     <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-
sasl'>
       <mechanism>EXTERNAL</mechanism>
       <mechanism>DIGEST-MD5</mechanism>
       <mechanism>PLAIN</mechanism>
       <required/></mechanisms></stream:features>
```

Klient čeká, dokud mu server nepošle features, jinak by nemohl začít zabezpečenou komunikaci. Poté klient čte seznam mechanismů šifrování a kontroluje, zda je mezi nimi mechanismus Digest-MD5 (u většiny serverů zpravidla bývá), který klient podporuje. Pokud ano, pošle klient oznámení, že si vybral kódování pomocí tohoto mechanismu.

XML element:	Kód programu:
`<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='DIGEST-MD5'/>`	Metoda: dataReceived(Object data) - step 2 ze třídy TLSDigestMD5Login.java

Server pošle challenge jehož tělo je zakódované zvolenou metodou (tedy metodou Digest-MD5):

```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
cmVhbG09InNvbWVyZWFsbSIsbm9uY2U9Ik9BNk1HOXRFUUdtMmhoIixxb
3A9ImF1dGgiLGNoYXJzZXQ9dXRmLTgsYWxnb3JpdGhtPW1kNS1zZXNzCg
==
</challenge>
```

Dekódované tělo challenge:

```
realm="somerealm",nonce="OA6MG9tEQGm2hh",qop="auth",chars
et=utf-8,algorithm=md5-sess
```

Význam jednotlivých tagů:

realm – nepovinný řetězec, který umožňuje uživateli rozpoznat které uživatelské jméno a heslo je používané

nonce – povinný řetězec vytvářený serverem, který slouží jako součást ověřování, je vhodné, aby byl náhodně generovaný a unikátní.

qop – nepovinný řetězec, který prozradí, jakou kvalitu ochrany nabízí server, může nabývat tří hodnot (auth - označuje základní ověření, auth-int - označuje autentifikaci s ochranou integrity a poslední auth-conf – označuje autentifikaci, ochranu integrity a šifrování). Pokud není hodnota uvedena, je výchozí hodnotou auth.

charset – nepovinný řetězec. Pokud je přítomen, oznamuje, že server podporuje kódování UTF-8 pro uživatelské jméno a heslo. Není-li přítomen, uživatelské jméno a heslo musí být kódovány v ISO 8859-1 (podmnožina US-ASCII).

algorithm mechanismus - povinný řetězec, určující způsob kódování

Na zprávu challenge obdrženou ze serveru je třeba, aby klient odpověděl XML elementem response. Response musí splňovat určité parametry a je vypočítána dle přesně daných postupů – viz níže.

XML element:

```
<response
xmlns='urn:ietf:params:xml:ns:xmpp-
sasl'>
dXNlcm5hbWU9InNvbWVub2RlIixyZWFsbT0ic29t
ZXJlYWxtIixub25jZT0iT0E2TUc5dEVRR20yaGgi
LGNub25jZT0iT0E2TUhYaDZWcVRyUmsiLG5jPTAw
MDAwMDAxLHFvcD1hdXRoLGRpZ2VzdC11cmk9Inht
cHAvZXhhbXBsZS5jb20iLHJlc3BvbnNlPWQzODhk
YWQ5MGQ0YmJkNzYwYTE1MjMyMWYyMTQzYWY3LGNo
YXJzZXQ9dXRmLTgK
</response>
```

Kód programu:

Metoda:

dataReceived(Object data) step 4

ze třídy TLSDigestMD5Login.java

Dekódované tělo response:

```
username="Myname",realm="somerealm",
nonce="OA6MG9tEQGm2hh",cnonce="OA6MHXh6VqTrRk",
nc=00000001,qop=auth,digest-uri="xmpp/example.org",
response=d388dad90d4bbd760a152321f2143af7,charset=utf-8
```

Vyplnění jednotlivých tagů:

username – uživatelské jméno, které se chce zalogovat na server

nonce – v odpovědi se použije stejná hodnota nonce, jako poslal server v elementu challenge.

cnonce – hodnota náhodně generovaná klientem

nc – hexadecimální číslo, které udává počet žádostí (a to včetně aktuální žádosti), které poslal klient se stejnou hodnotou nonce. Při první žádosti má tedy hodnotu 00000001.

response – počítá se dle vzorce:

response = hash (hash(A1),":"nonce,":"nc,":"cnonce,":"qop,":",hash(A2))

Kde:

hash = 16-bitový MD5 hash, 32-bitový řetězec hexadecimálních hodnot

A1 = hash(username, ":", realm, ":", passwd),":", nonce, ":", cnonce

A2 = "AUTHENTICATE:", digest-uri

Pokud je odpověď správně, pošle server ještě jednou challenge:

```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
 cnNwYXV0aD1lYTQwZjYwMzM1YzQyN2I1NTI3Yjg0ZGJhYmNkZmZmZAo=
</challenge>
```

Dekódované:

```
rspauth=ea40f60335c427b5527b84dbabcdfffd
```

Klient odpoví:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

V tomto momentě je daný uživatel úspěšně přihlášen k serveru. Server pošle informaci úspěšné autentizaci:

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Po úspěšné autentizaci se začíná vytvářet nová relace. Klient posílá úvodní hlavičku XML a vytvoří nový stream pro komunikaci směrem od klienta na server.

XML element:

```
<stream:stream
xmlns='jabber:client'
xmlns:stream='http://etherx.ja
bber.org/streams'
to='example.com'
version='1.0'>
```

Kód programu:

Metoda:

dataReceived(Object data) - step 5
ze třídy TLSDigestMD5Login.java

Na to server reaguje založením nového streamu pro komunikaci (směr ze serveru ke klientovi) a pošle doplňující seznam funkcí (nebo prázdný seznam, pokud se seznam podporovaných funkcí zabezpečeného spojení neliší od seznamu funkcí při nezabezpečeném spojení)

```
<stream:stream xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams'
id='c2s_456' from='example.com' version='1.0'>

<stream:features>
     <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
       <required/>
     </bind>
   </stream:features>
```

6.3.2 Roster

Roster neboli seznam kontaktů, je uchovávaný na serveru. Na serveru se ukládá proto, aby se mohl uživatel přihlásit pomocí jakéhokoli klienta a vždy dostal ten samý seznam. Poté co uživatel úspěšně přihlásí k serveru, zašle server požadavek na získání rosteru.

XML element:

```
<iq type='get' id='roster_123'>
      <query xmlns='jabber:iq:roster'/>
</iq>
```

Kód programu:

```
metoda:
getRoster()
ze třídy:
Roster.java
```

Server zašle seznam ve formátu:

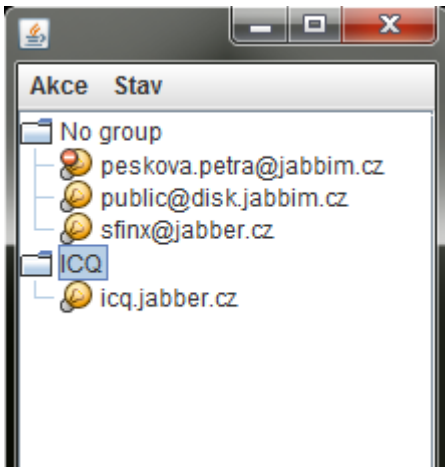
```
<iq type='result' id='roster_123'>
   <query xmlns='jabber:iq:roster'>
     <item jid='mujKamarad1@jehoServer.cz'
           name='Karel'
           subscription='both'>
       <group>Friends</group>
     </item>
     <item jid='mujKamarad2@jehoServer.com'
           name='Pepa'
           subscription='from'>
       <group>Friends</group>
     </item>
   </query>
 </iq>
```

V momentě, kdy program obdrží roster ze serveru, tak se pro každý objekt typu Item testuje, zda patří do nějaké skupiny či nepaří. Kontakty se poté ukládají do hash mapy `rosterData`, kde klíči jsou jednotlivé zjištěné skupiny a hodnoty klíče je seznam kontaktů, které do této skupiny patří. Hodnoty tedy tvoří list typu Item. Pokud kontakt nemá uvedenou žádnou skupinu, je přidělen do skupiny „NO_GROUP“. Jakmile jsou takto všechny kontakty roztřízeny, volá se metoda `updateRoster`, která se stará o vykreslení rosteru v hlavním okně programu.

Roster je zobrazován jako strom, je využito komponenty jTree. Strom se plní tak, že jednotlivé klíče hash mapy `dataRoster` jsou uzly stromu a seznam hodnot klíče je seznamem potomků daného uzlu.

Následující obrázek ukazuje, jakým způsobem je uživatel informován o stavu kontaktů, které má v rosteru. Stav kontaktu je indikován ikonou, umístěnou vlevo od jména.

Obrázek 6.1 Ukázka zobrazení rosteru




```
V případě kliknutí na kontakt pravým tlačítkem myši se
zobrazí nabídka pro smazání kontaktu. O to se stará
metoda deleteContact()

  public void deleteContact(String jid){
        Iq iq = new Iq();
        iq.setType("set");
        iq.setId("remove1");
        iq.setFrom(settings.getJid());
        Query query = new Query();
        Item item = new Item();
        item.setJid(jid);
        item.setSubscription("remove");
        query.getItem().add(item);
        iq.setAny(query);

        connection.sendData(iq);
    }
```

Metoda odesílá na server následující XML:

```
<iq type='set'
    from='ja@server.cz'
    id='remove1'>
  <query xmlns='jabber:iq:roster'>
    <item jid='kontakt1@server.com'
          subscription='remove'/>
  </query>
</iq>
```

Server na přijetí této žádosti reaguje následujícím způsobem:

1. Zašle kontaktu, kteráý mažeme žádost o odejmutí autorizace

```
<presence type='unsubscribe'
        from='ja@server.cz'
        to='Kontakt1@server.com'/>
```

2. poté zašle oznámení o zrušení autorizace z naší strany (to znamená, že odebíraný kontakt od této chvíle nebude dostávat informace o změně mého statusu)

```
<presence type='unsubscribed'
        from='ja@server.cz'
        to='Kontakt1@server.com'/>
```

3. zašle informaci, že jsem offline

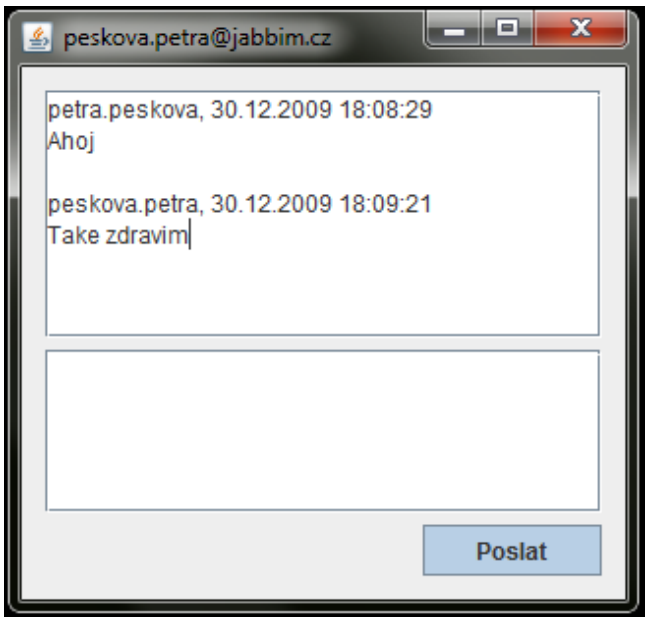
```
<presence type='unavailable'
        from='ja@server.cz/orchard'
        to='Kontakt1@server.com'/>
```

O přidání kontaktu do rosteru se zmiňuju v kapitole 6.3.4.

6.3.3 Zprávy

Zprávy se zobrazují v okně, jehož kód se nachází ve třídě MessageWin.java z balíku pogram.jabber.gui. Po dvojkliku na kontakt v rosteru se vytvoří nová instance této třídy. Všechny instance této třídy se uchovávají v hash mapě listOfMsgWin, která uchovává dvojice: JID – instance třídy MessageWin.java. Takže pokud například uživatel zavře okno s konverzací s jedním kontaktem a poté mu ještě tento kontakt odpoví, zobrazí se odpověď v původním okně i s celou historií konverzace.

Obrázek 6.2 Ukázka okna s konverzací



V názvu okna je zobrazeno JID kontaktu, kterému se zpráva odesílá. Horní JTextArea zobrazuje jak odeslané, tak i přijaté zprávy. Samotnou zprávu vždy předchází řádek se jménem pisatele zprávy a času. Dolní JTextArea je editovatelná a je určená pro psaní zprávy. Zpráva se odesílím kliknutím na tlačítko „Poslat“.

Zpráva se odesílá ve formátu:

```
<message to='mujKamard@jehoServer.cz' from='ja@neco.cz'>
          <body>Nechces jit na pivo?</body>
</message>
```

O takový formát se postará kód:

```
Message m = new Message();
m.setTo(„mujKamard@jehoServer.cz“);
m.setFrom(„ja@neco.cz“)
Body b = new Body();
b.setValue(„Nechces jit na pivo?“);
m.getAny().add(b);
messenger.sendMessage(m);
```

Nejprve se vytvoří nová instance třídy Message.java (generovaná z XSD schématu, nachází se v balíčku jaber.client). Pomocí metod, které jsou v této třídě implementované, se nastaví atributy elementu message. Element body má svou vlastní třídu. Pro připojení těla ke zprávě se využívá metody getAny(), pomocí které můžeme připojovat mimo elementu body i elementy subjekt či thread. Nakonec se volá metoda sendMessage(Message m) ze třídy Messenger pro odeslání výsledného XML na server.

Detekování příchozí zprávy probíhá ve třídě Messenger.java, kde metoda dataReceived(Object data) kromě testování, zdali jsou příchozí data ze serveru typu Message také předává zprávu třídě, která se stará o zobrazení hlavního okna aplikace. Tady probíhá další zpracování zprávy – testovaní, zda již bylo otevřeno okno s odesílatelem zprávy a pokud ne, vytvoří se nová instance třídy MessageWin, která příchozí zprávu zobrazuje.

```
@Override
public void dataReceived(Object data) {
  if (data instanceof Message) {
          Message m = (Message) data;
          if (m.containsBody()) {
                  gui.messageReceived(m);
           }
   }
}
```

6.3.4 Presence

Program se stará jednak o nastavení a zobrazení uživatele, a také zjišťuje a aktualizuje presenci všech kontaktů, které má uživatel v rostreru.

První odeslání informace o presenci uživatele se uskuteční ihned po úspěšném přihlášení uživatele na server. Nastavení presence uživatele má na starost metoda setPresence(PresenceMode mode, String message,byte priority).

```
/*  hlavička metody setPresence
 *  parametry: mode – nabývá hodnot online/away/dnd/unavailable
 *             message – obsahuje zprávu rozšířeného statusu
 *             priority – číselná hodnota udávající prioritu
 */
1   public void setPresence(PresenceMode mode, String message,
                            byte priority){
/*  vytvoření nové instance generované třídy presence.java
    z balíčku jabber.client*/
2   Presence presence = new Presence();
//  vytvoření elementu pro zobrazení statusu s hodnotou chat
3   JAXBElement<String> showElement = (new
    jabber.client.ObjectFactory()).createShow(„chat“);
/* vytvoření elementu pro nastavení priority presence
    s hodnotou 5
4   JAXBElement<Byte> priorityElement = (new
    jabber.client.ObjectFactory()).createPriority(5);
/* instance třídy Status, slouží pro zobrazení rozšíření
    Statusu */
5   Status statusElement = new Status();
//  nastavení hodnoty elementu status
6   statusElement.setValue(„Mám dobrou náladu. Vesele pište!“);
//  nastavení elementu priority stanze presence
7   presence.getAny().add(priorityElement);
//  nastavení elementu show stanze presence
8   presence.getAny().add(showElement);
//  nastavení elementu status stanze presence
9   presence.getAny().add(statusElement);
//  odeslání presence na server
10  connection.sendData(presence); }
```

Toto odešle na server následující XML dokument, který ohlašuje, že můj aktuální status je CHAT a sděluji ostatním „Mám dobrou náladu. Vesele pište“ s prioritou 5.

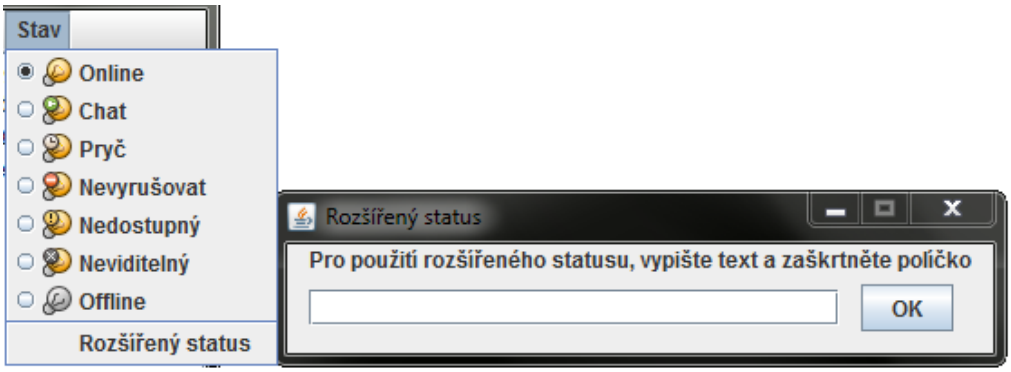
```
<presence>
  <show>chat</show>
  <status>Mám dobrou náladu. Vesele pište!</status>
  <priority>5</priority>
</presence>
```

Pozn.: Na řádku 3 a 4 vytvářím nové elementy (show a priority), protože generované třídy ze standardních XSD schémat tyto elementy neobsahují.

Uživatel může změnit svůj status kdykoli při běhu programu. Jednak je tu možnost změny na jeden ze statusů, které se přepínají v menu stav, a dále si

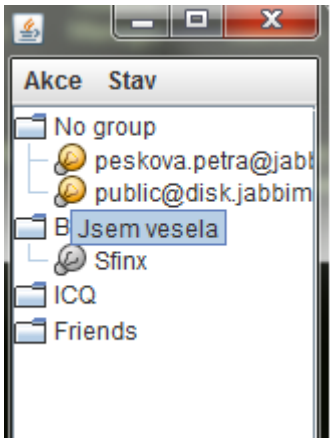
uživatel může sám napsat text rozšířeného statusu. Okno pro napsání vlastního rozšířeného statusu se zobrazí po kliknutí na položku „Rozšířený status“ v menu Stav.

Obrázek 6.3 Ukázka menu pro změnu statusu uživatele



Jakmile kdokoli ze seznamu kontaktů změní svůj status, obdrží klient od serveru zprávu o novém statusu. Na to program reaguje překreslením ikony u kontaktu, která informuje o jednom z možných stavů. Rozšířený status u kontaktů v rosteru zobrazuje po najetí kurzoru myši na kontakt tooltip.

Obrázek 6.4 Ukázka tooltipu zobrazujícího rozšířený status



Pokud uživatel přidává nový kontakt do svého rosteru, volí menu Akce a položku „Přidat kontakt“. Po té se zobrazí okno, kam je třeba napsat JID kontaktu, který se přidává.

Obrázek 6.5 Ukázka přidání nového kontaktu do rosteru

O následné přidání se stará metoda addContact(String jid).

```
1  public void addContact(String jid) {
2     Presence presence = new Presence();
3     presence.setType("subscribe");
4     presence.setFrom(settings.getJid());
5     presence.setTo(jid);
6     connection.sendData(presence);
    }
```

Nejprve se vytvoří nová instance třídy Presence z balíku jabber.client a poté se nastaví hodnoty objektů (viz výše) této třídy tak, aby se po použití marshallingu odesílalo na server následující XML:

```
<presence type='subscribe'
          from='ja@mujserver.com'
          to='mujKamarad@server.cz'/>
```

Uživateli, kterého přidáváme do rosteru, se po odeslání tohoto požadavku zobrazí informace o tom, že ho žádáme o potvrzení autorizace. V případě, že bude souhlasit, přijde od něj informace o potvrzení autorizace a zároveň žádost, abychom jej také autorizovali.:

```
<presence type='subscribed'
          from='mujKamarad@server.cz'
          to='ja@mujserver.com'/>
<presence type='subscribe'
          from='mujKamarad@server.cz'
          to=''ja@mujserver.com'/>
```

Poté nám server pošle informaci o stavu přidaného uživatele.

Pokud by přidávaný uživatel s autorizací nesouhlasil, oznámilo by to následující XML:

```
<presence type='unsubscribed'
          from='mujKamarad@server.cz'
          to=''ja@mujserver.com'/>
```

V takovém případě nebude klient informován o stavu přidávaného uživatele.

6.4 Testování programu

Program byl testován v operačních systémech Windows XP, Windows 7 a Ubuntu Karmic 9.10 (s grafickým prostředím KDE 4.3.4). Komunikace byla testována s klienty: Kopete 0.80.2 a Psi 0.13. Pro běh programu je zapotřebí mít nainstalovanou Javu 1.6 nebo vyšší.

Byly testovány se následující akce:

- autentizace uživatele na serverech jabber.cz a jabbim.cz – autentizace proběhla v pořádku, v případě v překlepu v hesle či uživatelském jménu nebo při potíží ze sítí, je program ošetřen a upozorní uživatele chybovou hláškou
- odeslání zprávy – proběhlo v pořádku, zprávy odeslané mým klientem se správně zobrazili v obou klientech pro testování
- přijetí zprávy – klient v pořádku přijímá zprávy od obou klientů (Psi i Kopete), avšak při testu výměny zpráv s webovým klientem Meebo.com došlo k chybě, klient Meebo.com posílá zprávy formátované pomocí xhtml standardu, aby je můj klient dokázal takovou zprávu zpracovat, byla by zapotřebí spolupráce s Java třídou generovanou ze schémat jabber.org/protocol/xhtml-im, vzhledem

k tomu, že tyto schémata nejsou validní, není možné třídu vygenerovat

- přidání kontaktu do rosteru a následné zasílání žádosti o autorizaci, po přidání kontaktu do rosteru klienti Psi i Kopete, detekovali autorizační žádost od mého klienta, kontakt se v pořádku uložil na server, po opětovném spuštění programu se zobrazuje
- smazání kontaktu – po smazání kontaktu, zmizí kontakt z rosteru (jak na serveru, tak v zobrazení klienta), smazaný uživatel dostane zprávu, že byl odstraněn a vidí uživatele mého klienta už jen offline
- reakce na autorizaci – po zaslání autorizační žádosti klienty Psi i Kopete, můj klient detekoval a zobrazil tuto žádost, naopak při akci smazání uživatele mého klienta v programu Psi nebo Kopete, reaguje můj klient informační zprávou, že jeho účet kontakt odstranil
- reakce na změnu presence kontaktu – pokud došlo ke změně statusu kontaktu, klient změnu detekoval a zobrazil (jak status, tak i rozšiřující zprávu)
- změna statusu – pokud uživatel změnil svůj status v mém klientu, byla tato změna v obou klientech Psi i Kopete vidět

7 Zhodnocení výsledků a závěr

Teoretická část práce podává stručný přehled protokolů, na kterých stojí komunikace v nejběžnějších komunikačních sítích. Jsou to protokoly: IRC, Gadu-Gadu, MSNP, YMSG, OSCAR, a XMPP.

Stěžejní částí práce je tvorba vlastního klienta pro komunikaci pomocí protokolu Jabber. Cílem této práce bylo vytvořit takové klienta, který splňuje následující podmínky:

- měl by disponovat základními funkcemi komunikátoru, jako jsou odesílání a přijímání textových zpráv, zobrazení stavu kontaktů z rosteru, modifikace vlastního statusu
- jednoduchý a nenáročný
- nezávislý na platformě

Výsledkem této práce je takový klient, cíle práce byly splněny. Díky vytvořenému GUI je program jednoduše ovladatelný. Vzhledem k tomu, že program byl vyvíjen v jazyce Java, není závislý na platformě. Splňuje všechny funkce, které byly v kapitole 2 stanoveny.

Klient dokáže navázat TCP/IP spojení na portu 5222 s libovolným Jabber serverem. Poté dojde k vytvoření šifrovaného kanálu nad tímto spojením pomocí TLS protokolu. Při odesílání přihlašovacích údajů se používá šifrovací mechanismus SASL-MD5, heslo se tedy nikdy neposílá po síti jako holý text. Pokud tento proces neproběhne v pořádku, je uživatel informován chybovou hláškou, upozorňující na chybu sítě nebo chybné přihlašovací údaje. Poté je mu nabídnuto přihlášení zopakovat.

Po zkušebních testech (viz předchozí kapitola) se ověřilo, že skutečně dokáže odesílat a přijímat textové zprávy. Ty jsou zobrazovány

v komunikačním okně. Pro každý kontakt, se kterým klient komunikuje, je zobrazeno právě jedno komunikační okno. Pokud je toto okno zavřeno a poté ještě přijde od kontaktu zpráva (nebo pokud se uživatel rozhodne ještě něco napsat), je obnoveno původní okno i s historií chatu.

Dále si klient umí od serveru vyžádat zaslání rosteru, ten následně zpracovat - zobrazit jako strom, kontakty se třídí dle skupiny, do které patří. Roster je možné modifikovat, ve smyslu přidání či smazání kontaktu. Při přidávání kontaktu se posílá žádost o autorizaci, pokud si naopak někdo přidává nás do rosteru, program na to reaguje zobrazením okna s dotazem, zda s autorizací souhlasíme. Při smazání kontaktu se odebírá autorizace, aby do budoucna nebyl smazaný kontakt informován o statusu uživatele.

Klient dokáže zpracovat statusy kontaktů, které má uložené v rosteru. Jednak zobrazuje stav kontaktu (zobrazením ikony vedle jména kontaktu) a dále zobrazuje i zprávu rozšiřující popis stavu (po najetí myši zobrazí tooltip, ve kterém je rozšiřující zpráva).

Změna statusu přihlášeného uživatele je také funkční, uživatel může přepínat mezi statusy online, volný pro chat, pryč, nevyrušovat, nedostupný a offline. Dále je možné nastavit rozšiřující zprávu statusu.

Na závěr nastíním další možnosti rozvoje programu. Myslím, že zajímavá možnost vylepšení programu by byla inovace vzhledu, přidání podpory emoikon a obohacení programu o zvukové upozornění na příchozí zprávy.

Reference

[1] KEVINLE. *Live Services* [online]. March 18, 2009 [cit. 2009-03-19]. Dostupný z WWW: <http://dev.live.com/blogs/devlive/archive/2009/03/18/481.aspx>.

[2] Microsoft Corporation. Internet-Draf : Instant Messaging and Presence Protocol [online]. 1999 [cit. 2009-03-03]. Dostupný z WWW: <http://www.hypothetic.org/docs/msn/ietf_draft.txt>.

[3] OIKARIN, J., REED, D.. *RFC1459 - Internet Relay Chat Protocol* [online]. May 1993 [cit. 2009-10-02]. Dostupný z WWW: <http://www.faqs.org/rfcs/rfc1459.html>.

[4] DOČEKAL, Daniel. *Orey Gilliam: ICQ již nevidí krádeže účtu jako velký problém* [online]. 2. 7. 2008 6:30 [cit. 2009-08-06]. Dostupný z WWW: <http://www.lupa.cz/clanky/orey-gilliam-icq-nevidi-kradeze-uctu-jako-problem/>.

[5] Polskie Badania Internetu [online]. Warszawa: c2005 [cit. 2009-03-02]. Text v polštině. Dostupný z WWW: <http://panel.pbi.org.pl/megapanel.php>.

[6] P. SAINT-ANDRE, ED.. *Extensible Messaging and Presence Protocol (XMPP): Core* [online]. October 2004 [cit. 2009-07-10]. Dostupný z WWW: <http://www.ietf.org/rfc/rfc3920.txt>.

[7] P. SAINT-ANDRE, ED.. *Extensible Messaging and Presence Protocol (XMPP) : Instant Messaging and Presence* [online]. October 2004 [cit. 2009-07-02]. Dostupný z WWW: <http://www.ietf.org/rfc/rfc3921.txt>.

[8] W3C/MIT. *Hypertext Transfer Protocol -- HTTP/1. : 10 Status Code Definitions* [online]. June 1999 [cit. 2009-09-08]. Dostupný z WWW: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>.

[9] XML Stream Features [online]. 2003 , 2009-06-17 [cit. 2009-10-31]. Dostupný z WWW: <http://xmpp.org/registrar/stream-features.html>.

Použitá literatura

ADAMS, Dj. *Programming Jabber : Extending XML Messaging*. [s.l.] : O\'Reilly Media, 2002. 480 s. Dostupný z WWW: <http://commons.oreilly.com/wiki/index.php/Programming_Jabber>.

AOL. *OAL Developer Network : OSCAR Protocol* [online]. on March 5, 2008 - 2:38am. [cit. 2009-10-12]. Dostupný z WWW: <http://dev.aol.com/aim/oscar>.

Google. *Google Talk for Developers : Open Communications* [online]. c2009 [cit. 2009-10-24]. Dostupný z WWW: <http://code.google.com/intl/cs-CZ/apis/talk/open_communications.html>.

IRC .diary.cz : Základy IRC Sítí [online]. - [cit. 2009-09-14]. Dostupný z WWW: <http://irc.diary.cz/network.html>.

MEUNIER, Frédéric L. W.. *Linux IRC mini-HOWTO* [online]. Fifth revision. January, 2005 , 2005-01-07 [cit. 2009-09-14]. Dostupný z WWW: <http://tldp.org/HOWTO/IRC/index.html>.

MILLER, Matthew , et al. *XEP-0070: : Verifying HTTP Requests via XMPP* [online]. Version 1.0. c1999-2009 , 2005-12-14 [cit. 2009-09-22]. Dostupný z WWW: <http://xmpp.org/extensions/xep-0070.html>.

SAINT-ANDRE, Peter, et al. *XEP-0100: : Gateway Interaction* [online]. c1999-2009 [cit. 2009-08-08]. Dostupný z WWW: <http://xmpp.org/extensions/xep-0100.html>.

SCOTT , Ludwig, et al. *XEP-0166: Jingle* [online]. c1999-2009 , 2009-12-23 [cit. 2009-10-22]. Dostupný z WWW: <http://xmpp.org/extensions/xep-0166.html>.

SHUTKO, Alexandr . *OSCAR (ICQ v7/v8/v9) protocol documentation* [online]. - , 07.02.2005 [cit. 2009-10-12]. Dostupný z WWW: <http://iserverd.khstu.ru/oscar/>.

Seznam příloh:

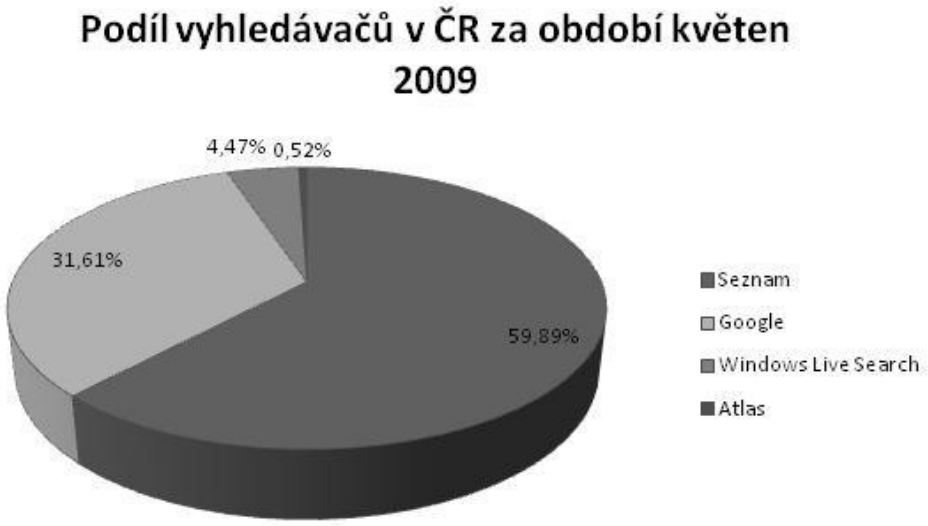
A. Data pro výběr protokolů

Tabulka 2.2.1:

Komunikační síť	Vyhledávaný výraz	Počet vyhledaných výskytů v česky psaných článcích			
		Seznam	Google	Windows Live Search	Celkem
Bonjour	Bonjour protokol	777	1430	183	2 390
Gadu-Gadu	Gagu gadu	2035	10 700	2080	14 815
Google Talk	Google Talk	63950	39 900	34000	137 850
Groupwise	Groupwise	1643	6220	1190	9 053
IAX	IAX protokol	335	1060	260	1 655
ICQ	ICQ	16449422	3990000	1 100 000	21 539 422
IRC	IRC	388 205	301 000	211 000	900 205
MSN	MSN	2056369	798000	24 500	2 878 869
MySpaceIM	MySpaceIM	124	582	143	849
Netsend	netsend	215	2450	179	2 844
QQ	Tencent QQ	61	528	94	683
Sametce	Sametime protokol	199	2 970	107	3 276
Skype	Skype	7305208	1520000	14 000	8 839 208
SILC	Silc protokol	171	660	112	943
Tlen	Tlen protokol	379	644	89	1 112
XFIRE	Xfire protokol	196	967	112	1 275
XIMSS(SIP)	Ximss sip	99	370	85	554
XMPP	Jabber	516 153	370 000	21 700	907 853
Yahoo!	Yahoo! Messenger	437 768	320 000	75 300	833 068
Zephyr	Zephyr protokol	239	398	121	758

Tabulka názorně ukazuje počet vyhledaných česky psaných článků na téma daného komunikačního protokolu. Tyto články jsem vyhledávala v prvních třech nejpoužívanějších prohlížečích v ČR. Žebříček prohlížečů blíže popisuje graf č. 2.2.2. Z tabulky je vidět, že mezi nejznámější komunikační sítě v ČR patří: ICQ, MSN, XMPP, IRC, Yahoo!, Google Talk a Gadu-Gadu.

Graf č. 2.2.2



Zdroj: http://www.iinfo.cz/tiskove-centrum/tiskove-zpravy/navrcholu-vyhledavace-kveten/

Tabulka 2.2.2

Nesjtahovanější klienti ze serveru stahuj.cz za období posledního roku:

Název klienta	Počet stažení	Podporované protokoly
ICQ	891922	ICQ
QIP 2005	658883	ICQ
QIP Infium	167484	Jabber a XIMSS(SIP)
QIP Infium čeština + protokoly	97485	Jabber, ICQ, XIMSS (SIP)
Miranda	54919	ICQ, IRC, AIM, MSN, Skype, Jabber, Yahoo! Messenger, Gadu-Gadu, Tlen, Netsend
QIP Infium JadrisPack	48920	ICQ, Jabber, XIMSS, IRC
Windows Live Messenger	45177	MSN
Wolfram3D Miranda-Pack	38539	ICQ
Miranda IM KenDASS BLACK pack	33239	ICQ, Jabber a MSN
QIP Infium PafoPack	22507	ICQ, Jabber

B. Obsah přiloženého CD

Na přiloženém jsou složky Text a Program. Ve složce Text je uložen pdf dokument bakalarskaPrace.pdf, kde je uveden celý text této práce. Ve složce Program se nachází program jednak ve formě NetBeans projektu, dále jako jar soubor a spustitelný bat soubor.